Turning the Coq Proof Assistant into a Pocket Calculator

Guillaume Melquiond

Université Paris-Saclay, Inria, LMF

September 14, 2024

The Need for Guaranteed Mathematical Computations

Harald Helfgott on MathOverflow, 2013 (link)

I need to evaluate some (one-variable) integrals that neither SAGE nor Mathematica can do symbolically. As far as I can tell, I have two options:

a. Use GSL (via SAGE), Maxima or Mathematica to do numerical integration. This is really a non-option, since, if I understand correctly, the "error bound" they give is not really a guarantee.

b. Cobble together my own programs using the trapezoidal rule, Simpson's rule, etc., and get rigorous error bounds using bounds I have for the second (or fourth, or what have you) derivative of the function I am integrating. This is what I have been doing.

Is there a third option? Is there standard software that does (b) for me?

The Need for Guaranteed Mathematical Computations

Harald Helfgott on MathOverflow, 2013 (link)

I need to evaluate some (one-variable) integrals that neither SAGE nor Mathematica can do symbolically. As far as I can tell, I have two options:

a. Use GSL (via SAGE), Maxima or Mathematica to do numerical integration. This is really a non-option, since, if I understand correctly, the "error bound" they give is not really a guarantee.

b. Cobble together my own programs using the trapezoidal rule, Simpson's rule, etc., and get rigorous error bounds using bounds I have for the second (or fourth, or what have you) derivative of the function I am integrating. This is what I have been doing.

Is there a third option? Is there standard software that does (b) for me?

The software suggested by the accepted answer computes an incorrect value on the example proposed by Helfgott.

Computing in Coq

One of the strengths of Coq is its ability to compute

```
Compute (3 + 5)%Z. (* = 8 : Z *)
```

Computing in Coq

One of the strengths of Coq is its ability to compute

```
Compute (3 + 5)%Z. (* = 8 : Z *)
```

But there are a few shortcomings

- No immediate relation between the input and the result.
- Works poorly with abstract symbols:

```
Compute (3 + 5)%R.
(* = R1 + (R1+R1) + (R1 + (R1+R1) * (R1+R1)): R *)
```

Computing in Coq

One of the strengths of Coq is its ability to compute

```
Compute (3 + 5)%Z. (* = 8 : Z *)
```

But there are a few shortcomings

- No immediate relation between the input and the result.
- Works poorly with abstract symbols:

```
Compute (3 + 5)%R.
(* = R1 + (R1+R1) + (R1 + (R1+R1) * (R1+R1)): R *)
```

Yet, if the result is known, one can do a formal proof

```
Goal (3 + 5 = 8)%R. Proof. ring. Qed.
```

Coq as a Pocket Calculator

Objectives

- Leverage the proof system of Coq.
- Give some meaningful answers to the user.
- Make it user-friendly.

Outline

Existential Variables and Tactic-in-Terms

$$3 + 5?$$

Leaving holes in the goal: existential variables

```
eassert (3 + 5 = ?[r]) as H.
{ (* 3 + 5 = ?r *)
  ring_simplify. reflexivity. }
(* H: 3 + 5 = 8 |- ... *)
```

Existential Variables and Tactic-in-Terms

$$3 + 5?$$

Leaving holes in the goal: existential variables

```
eassert (3 + 5 = ?[r]) as H.
{ (* 3 + 5 = ?r *)
  ring_simplify. reflexivity. }
(* H: 3 + 5 = 8 |- ... *)
```

Leaving the whole goal as a hole: tactic-in-terms

```
Definition foo := ltac:(
  refine (_ : 3 + 5 = _) ;
  ring_simplify ; reflexivity).
Check foo. (* foo : 3 + 5 = 8 *)
```

A Rough Calculator

Tying things together

```
Ltac expand t := refine (_: (t = _));
  ring_simplify; reflexivity.

Definition foo := ltac:(expand (3 + 5)).
Check foo. (* 3 + 5 = 8 *)
```

A Rough Calculator

Tying things together

```
Ltac expand t := refine (_: (t = _));
  ring_simplify; reflexivity.

Definition foo := ltac:(expand (3 + 5)).
Check foo. (* 3 + 5 = 8 *)
```

Unfriendly, but actually powerful

```
Definition bar x := ltac:(expand ((x+1) * (x-1))).
Check bar. (* forall x, (x+1) * (x-1) = x^2 - 1 *)
```

Interval Arithmetic in a Nutshell

Naive interval arithmetic

If $u \in [\underline{u}; \overline{u}]$ and $v \in [\underline{v}; \overline{v}]$,
then $u - v \in [\underline{u} - \overline{v}; \overline{u} - \underline{v}]$.

Interval Arithmetic in a Nutshell

Naive interval arithmetic

If $u \in [\underline{u}; \overline{u}]$ and $v \in [\underline{v}; \overline{v}]$,
then $u - v \in [\underline{u} - \overline{v}; \overline{u} - \underline{v}]$.

Rigorous polynomial approximations: $f \in \langle P_f, \Delta_f \rangle_X$

If $u(x) - P_u(x) \in \Delta_u$ and $v(x) - P_v(x) \in \Delta_v$ for all $x \in X$,
then $(u - v)(x) - (P_u - P_v)(x) \in \Delta_u - \Delta_v$ for all $x \in X$.

Interval Arithmetic in a Nutshell

Naive interval arithmetic

If $u \in [\underline{u}; \overline{u}]$ and $v \in [\underline{v}; \overline{v}]$,
then $u - v \in [\underline{u} - \overline{v}; \overline{u} - \underline{v}]$.

Rigorous polynomial approximations: $f \in \langle P_f, \Delta_f \rangle_X$

If $u(x) - P_u(x) \in \Delta_u$ and $v(x) - P_v(x) \in \Delta_v$ for all $x \in X$,
then $(u - v)(x) - (P_u - P_v)(x) \in \Delta_u - \Delta_v$ for all $x \in X$.

The CoqInterval library

```
Goal forall x, 0 <= x <= 1 -> sin (x + exp x) = 0 ->
     0.835 <= x <= 0.836.
Proof. intros x H E. root E. Qed.
```

Leveraging CoqInterval's Tactics

Interval arithmetic as a proof helper

```
interval_intro (PI^2/6) with (i_prec 10) as H.
(* H: 841/512 <= PI^2/6 <= 844/512 |- ... *)
```

Leveraging CoqInterval's Tactics

Interval arithmetic as a proof helper

```
interval_intro (PI^2/6) with (i_prec 10) as H.
(* H: 841/512 <= PI^2/6 <= 844/512 |- ... *)
```

Leveraging the tactics

```
Definition foo := ltac:(
  let H := fresh in
  interval_intro (PI^2/6) with (i_prec 10) as H;
  exact H).
Check foo. (* 841/512 <= PI^2/6 <= 844/512 *)
```

Leveraging CoqInterval's Tactics

Interval arithmetic as a proof helper

```
interval_intro (PI^2/6) with (i_prec 10) as H.
(* H: 841/512 <= PI^2/6 <= 844/512 |- ... *)
```

Leveraging the tactics

```
Definition foo := ltac:(
  let H := fresh in
  interval_intro (PI^2/6) with (i_prec 10) as H;
  exact H).
Check foo. (* 841/512 <= PI^2/6 <= 844/512 *)
```

Making the tactics recognize goal evars

```
Definition foo :=
  ltac:(interval (PI^2/6) with (i_prec 10)).
```

Outline

1. Introduction
2. A rough calculator
3. Improving the user experience
   - Vernacular commands to the rescue
   - Postprocessing
4. Conclusion

Vernacular Commands to the Rescue

Objectives

1. Have a single short command per user query.
2. Make the proof term opaque.
3. Postprocess the type of the proof term.
4. (Improve performance.)

Vernacular Commands to the Rescue

Objectives

1. Have a single short command per user query.
2. Make the proof term opaque.
3. Postprocess the type of the proof term.
4. (Improve performance.)

Def and Do

```
Do interval (PI^2/6).
(* (PI ^ 2 / 6) ≃ 1.64493406685 *)
Def foo x ‘(0 <= x <= 1) := root (sin (x + exp x)).
(* x ≃ 0.835538085216 *)
```

Postprocessing the Proof Type

Why postprocess?

```
Def foo x ‘(0 <= x <= 1) := root (sin (x + exp x)).
(* x ≃ 0.835538085216 *)

Check foo.
(* forall x, 0 <= x <= 1 -> sin (x + exp x) = 0 ->
   7525858018462367 / 9007199254740992 <= x <=
   7525858018462401 / 9007199254740992 *)
```

Postprocessing the Proof Type

Why postprocess?

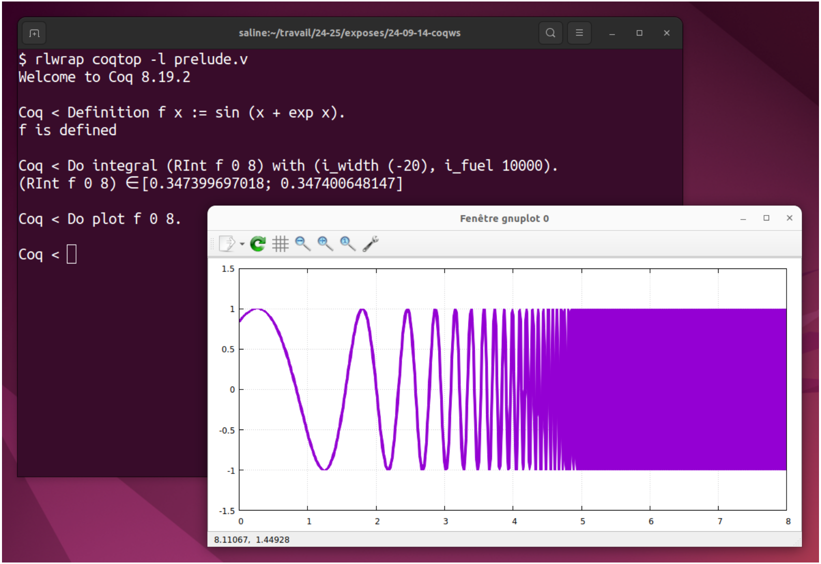
```
Def foo x `(0 <= x <= 1) := root (sin (x + exp x)).
(* x ≃ 0.835538085216 *)

Check foo.
(* forall x, 0 <= x <= 1 -> sin (x + exp x) = 0 ->
   7525858018462367 / 9007199254740992 <= x <=
   7525858018462401 / 9007199254740992 *)
```

What about wide intervals?

```
Do integral (RInt (fun x => sin (x + exp x)) 0 8)
   with (i_width (-20), i_fuel 1000).
(* (RInt (fun x : R => sin (x + exp x)) 0 8)
   ∈ [0.347399697018; 0.347400648147] *)
```

Types are More Than Just Text



Outline

The Issue With Performance

Computations are performed thrice

```
Definition slow n := (fact n) mod (S n).

Ltac reduce v :=
  let w := eval vm_compute in v in
  exact_no_check (eq_refl w <: v = w).

Time Do reduce (slow 12). (* 102.7s *)

Time Eval vm_compute in slow 12. (* 34.2s *)
```

The Issue With Performance

Computations are performed thrice

```
Definition slow n := (fact n) mod (S n).

Ltac reduce v :=
  let w := eval vm_compute in v in
  exact_no_check (eq_refl w <: v = w).

Time Do reduce (slow 12). (* 102.7s *)

Time Eval vm_compute in slow 12. (* 34.2s *)
```

Naming expressions triggers memoization

```
Definition aux := slow 12.
Time Do reduce aux. (* 31.0s *)
```

Work in progress: Teach `Do`, `Def`, and the tactics how to create intermediate definitions.

Conclusion

The `Do` & `Def` commands

1. Invoke a given tactic to compute some proof term.
2. Postprocess and display the type of the proof term.

Conclusion

The Do & Def commands

1. Invoke a given tactic to compute some proof term.
2. Postprocess and display the type of the proof term.

"Is there standard software that does (b) for me?"

```
Do integral (RInt (fun x => Rabs (
        (x^4 + 10*x^3 + 19*x^2 - 6*x - 6) * exp x
      )) 0 1) with (i_relwidth 50).
(* (RInt (fun x => ...) 0 1) ≃ 11.1473105501 *)
```

Available in CoqInterval

https://coqinterval.gitlabpages.inria.fr/